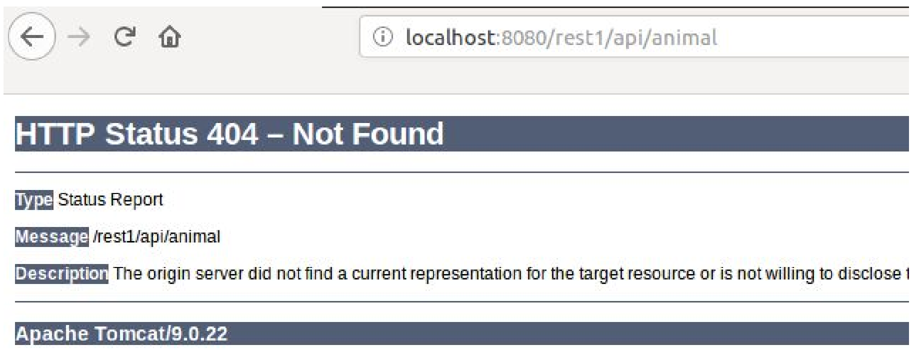
Restful web service - simple

Exercise 1: Simple Rest-endpoint

1. From within Netbeans make a new project: Java with Maven -> Web Application.
2. Project name: rest1
3. Server: Tomcat9
4. In Source Packages create a new package: `rest`
5. Right-click the new package → New → Other→ Web Services → “RESTful Web Services from Patterns” → next → Simple Root Resource → next →
    a. Path: animals
    b. Class Name: AnimalDemo
    c. MIME Type: application/json → finish
6. Note that 2 files were created by the previous steps:
    a. AnimalDemo.java
    b. ApplicationConfig.java
7. AnimalDemo.java is our actual web service file. Open it and see how both the class and its methods have annotations like:
    a. @Path("generic")
    b. @GET
    c. @Produces(MediaType.APPLICATION_JSON)
    d. @PUT
    e. @Consumes(MediaType.APPLICATION_JSON)
8. Now open `ApplicationConfig` and change “webresources” to “api” in line: @javax.ws.rs.ApplicationPath("webresources").
9. Note also how the method: `addRestResourceClasses` now creates a reference to our new web service: `AnimalDemo.class`
10. Back to `AnimalDemo.java`: Change the class annotation from “generic” to “animals”
11. Change the method annotated with `@GET`:
    a. Remove the statement: throw new …
    b. Change the Annotation: @Produces(MediaType.APPLICATION_JSON) to @Produces(MediaType.TEXT_PLAIN)
    c. Write statement: return "Vuf… (Message from a dog)";
12. Right-click the project and select Run. This opens the index.html page in a browser served by your Tomcat web server.
13. Change the URL to `localhost:8080/rest1/api/animals`



localhost:8080/rest1/api/animal

HTTP Status 404 – Not Found

Type Status Report

Message /rest1/api/animal

Description The origin server did not find a current representation for the target resource or is not willing to disclose t

Apache Tomcat/9.0.22

14. What went wrong? We are missing a required dependency for REST with Tomcat.
15. In Netbeans, right-click the Dependencies folder → add Dependencies → In “Query” write: **jaxrs-ri** → In “Search Results” select: **`jaxrs-ri`** (latest 2.x version)
16. Clean and Build the project
17. Run the project again, and in the browser go to `localhost:8080/rest1/api/animal`
    Verify that your web service is accessible (i.e. fetches the message)
18. Reflect on every step you took. You will need to do this many times this semester
    a. Summarize (in writing?) the crucial steps for making this work.

Exercise 2: Json endpoint

1. Now create another endpoint (another java method with a different @Path.)
    - Create a new java method
    - Above it create 3 annotations:
      ```
      @GET
      @Path(“/animal_list”)
      @Produces(MediaType.APPLICATION_JSON)
      ```
    - Let the method return a string like this: `"[\"Dog\", \"Cat\", \"Mouse\", \"Bird\"]"` (a JSON list of strings)
2. Save and go to `localhost:8080/animal/animal_list` to see the JSON result
3. Reload the page with the browsers network tab open
4. Click on the GET request and check the response headers. What is the Content-type?

Exercise 3: Return an object

1. In your project create a new package: `model`
2. Inside it create a java class: AnimalNoDB with two fields:
    a. type (Dog, Duck etc..)
    b. sound (Bark, Quack etc..)
3. Create a constructor that takes a type and sound
4. In your rest service file (AnimalDemo.java file) add a new rest-endpoint annotated like this:
   @GET
   @Path(“/animal”)
   @Produces(MediaType.APPLICATION_JSON)

5. In this method create an AnimalNoDB object and serialize it to json using the Gson library which you get like this: Add a new dependency: `com.google.code.gson:gson`
    - Remember to rebuild project after adding dependencies
    - Hint: `return new Gson().toJson(animal);`
    - Save and rerun project.
    - The result should be something like: {"type":"Duck","sound":"Quack"}

Exercise 4: Use JPA and a DataBase with your REST Endpoint

Create the EntityClass for this demo
Create a new package **`entity`**
In this package use the Wizard to create a new Entity Class called Animal. Make sure to select "create persistence.xml" and use **pu** as the name for Persistence Unit.
When you set up the *New Database Connection* just use the existing database `startcode` and the existing username **`dev`** and password **`ax2`**.

Add two string fields `type` and `sound` (as in the AnimalNoDB class in the model package)
Create a constructor that takes type and sound + the required no-args constructor.

Create the class for the new endpoints
You could continue and just add additional endpoints to the existing AnimalDemo class, but to separate concerns, we will create a new class for the following endpoints.

1) Use the wizard (again) as follows:
Right-click the `rest` package → New → Other→ Web Services → "RESTful Web Services from Patterns" → next → Simple Root Resource → next →

a. Path: animals_db
b. Class Name: AnimalFromDB
c. MIME Type: application/json → finish

2) Remove all the existing endpoints in the class, and add a static field at the top of the class to hold the EntityManagerFactory as sketched below:

```
private static EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");
```

3) Add this endpoint to the class and verify that you can access the endpoint (figure out how)

```
@Path("animals")
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getAnimals() {
  EntityManager em = emf.createEntityManager();
  try{
      TypedQuery<Animal> query = em.createQuery("SELECT a FROM Animal a", Animal.class);
      List<Animal> animals = query.getResultList();
      return new Gson().toJson(animals);
   } finally {
         em.close();
   }
}
```

If you have selected the **`create`** strategy in `persistence.xml` calling this endpoint should create the table Animal (if not already created), but since there is no data yet, it should return an empty JSON list like so: [ ]

4) Verify the existence of the table using Workbench.

5) Use Workbench to insert some test data like below:

```
insert into startcode.ANIMAL( type,sound) values
("Dog","VUF"),("DUCK","Quack"),("CAT","Miav");
```

Restart the project and verify that you get a JSON-list with all the animals (as objects) inserted above from the server.

Now it's your turn to THINK ;-)

Implement the following endpoints in the AnimalFromDB class.

`/animals_db/animalbyid/{id}`

This should return the animal found with the given id, or null if no animal with this id exists (You will learn better ways to handle the last scenario latter)

Hint: You will need the PathParam annotation for this one, check here for example(s) and example below:

```
@Path("animalbyid/{id}")
@GET
@Produces(MediaType.APPLICATION_JSON)
public String getAnimal(@PathParam("id") int id) {
  //Hvis den kaldes med .../animalbyid/2  vil id nu være lig 2.
  //Den værdi kan I så benytte til at slå op i databasen med em.find(
```

`/animals_db/animalbytype/{type}`

Should return the animal found with the given type, or null in no animal was found

Hint: You will need the PathParam as above, but the type has to be different. If not you will have two methods with the same name.

`/animals_db/random_animal`

Should return a random animal selected from what is available in the database or null if no animals exist.